

HTTP klijent i server

Za implementaciju HTTP servera u Pythonu ćemo koristiti ugrađeni modul `http.server`, dok ćemo za klijentsku stranu, umesto korišćenja browsera kao klijenta, koristiti ugrađeni modul `urllib3`. Pošto su ovi moduli pisani u objektno orijentisanom Pythonu, oni pružaju objektno orijentisan pristup implementaciji.

Serverske klase i klase za obradu

HTTP serveri koji se kreiraju u Pythonu pomoću `http.server` modula se sastoje iz dva dela:

- `HTTPServer` klasa – ugrađena je u sam modul i ista je za svaki server baziran na ovom modulu; omogućava nam osluškivanje po određenom portu, kao i prihvatanje zahteva od klijenta;
- klasa za obradu (handler class) – nakon što `HTTPServer` klasa primi zahtev, prosleđuje tu informaciju klasi za obradu (handler klasa) koju mi implementiramo.

Implementacije HTTP klijenta (`http_local_client.py`) i servera (`http_local_server.py`) ćemo prikazati na primeru gde server sadrži rečnik imena i prezimena nazvan `names_dict`, a klijent pokušava ili da dobavi prezime za dato ime (GET) ili da doda nove vrednosti tom rečniku (POST) koristeći upitne stringove.

HTTP server

Pošto za implementaciju HTTP servera koristimo `HTTPServer` klasu koja je deo `http.server` modula, uvešćemo je pomoću `from` – `import` sintakse. Takođe, naša klasa za obradu će nasledivati klasu `BaseHTTPRequestHandler`, koja je takođe deo ovog modula, pa ćemo i nju uvesti: `from http.server import HTTPServer, BaseHTTPRequestHandler`.

Za instanciranje `HTTPServer` objekta (objekta koji će nam služiti kao sam server) potrebne su nam dve stvari kao argumenti:

- n-torka koja sadrži dva elementa – IP adresu na kojoj se server nalazi i port (prema specifikaciji protokola, podrazumevani portovi u HTTP protokolu su 8080 i 80);
- ime klase za obradu koju ćemo naknadno definisati.

Instanciranje serverskog objekta se vrši linijom `http_server = HTTPServer(('127.0.0.1', 8080), RequestHandler)`. U našem primeru, tu klasu zaduženu za obradu koju ćemo implementirati smo nazvali `RequestHandler`. Pošto smo kreirali http serverski objekat, pokrenućemo ga pomoću `HTTPServer` metode `serve_forever()`.

U ovom trenutku naš HTTP server je pokrenut na lokalnoj IP adresi (127.0.0.1) i portu 8080 i po toj adresi i portu mu se može pristupiti. Pre pokretanja ovog fajla moramo definisati i klasu za obradu.

Klasa za obradu (handler class)

Samu klasu za obradu koja nasleđuje `BaseHTTPRequestHandler` klasu možemo definisati sa `class RequestHandler(BaseHTTPRequestHandler)`. Ono što nam nasleđivanje `BaseHTTPRequestHandler` klase omogućava jeste to da čim dobijemo zahtev od klijenta – taj zahtev ćemo usmeriti ka našoj odgovarajućoj metodi. Zato, imena metoda ove klase imaju tačan šablon po kojem se moraju nazivati: `do_GET`, `do_POST`, `do_PUT` itd. Pošto znamo da klijent, kada želi da dobavi određeni resurs ili informaciju sa HTTP servera, koristi HTTP metodu `GET`, nju ćemo prvo i implementirati. Dakle, u našoj klasi `RequestHandler`, metoda za obrađivanje `GET` zahteva će se zvati `def do_GET(self)`.

Metoda GET

Prilikom slanja odgovora na zahtev u našem primeru, naš HTTP server će poslati tri parametra:

- statusni kod;
- zaglavlje;
- podatak koji se traži (telo poruke (body)).

Takođe, ove parametre HTTP mora poslati i upravo tim redosledom. Pre slanja odgovora, naš HTTP server mora prvo analizirati i raščlaniti upitni string (query string) koji je došao sa `GET` zahtevom (kojeg je klijent poslao). Ovaj string je deo samog URL-a i izgleda ovako: `http://127.0.0.1:8080/?name=michael`. U ovom primeru URL-a, `http://127.0.0.1:8080/` je adresa na kojoj se server nalazi, dok je deo `name=michael` zapravo upitni string koji se sastoji iz dva dela:

- promenljiva koju tražimo – u našem slučaju je to `name` (ime);
- vrednost te promenljive – u našem slučaju je `michael`.

Karakter `?` je indikator i označava da od tog dela počinje upitni string.

Jedan URL, tačnije jedan `GET` zahtev može sadržati više ovakvih promenljiva–vrednost parova za upitne stringove razdvojenih ampersandom (`&`) u formi:

```
http://127.0.0.1:8080/?name=peter&last_name=peterson
```

U ovom slučaju smo prosledili dve promenljive: `name` i `last_name`, sa njihovim vrednostima.

Za ovakvo raščlanjivanje upitnih stringova u Pythonu postoji već ugrađena funkcionalnost koja je deo `urllib.parse` biblioteke, preciznije funkcija `parse_qs`, kojoj se ovakav string prosleđuje, a koja vraća rečnik:

Primer raščlanjivanja pomoću <code>parse_qs()</code> funkcije
Kod
<pre>from urllib.parse import parse_qs print(parse_qs('name=peter&last_name=peterson'))</pre>

Tabela 8.1. Primer raščlanjivanja pomoću `parse_qs()` funkcije

Kao što vidite, rezultat je sledeći:

```
{'name': ['peter'], 'last_name': ['peterson']}
```

Dakle, prvo što server treba da odredi jeste statusni kod zahteva (da li se radi o uspešnom zahtevu, klijentskoj ili serverskoj grešci ili drugo). Lista svih statusnih kodova HTTP protokola i poruka koje oni pokazuju može se pronaći [ovde](#). Slanje statusnog koda činimo metodom nasleđenom od BaseHTTPRequestHandler klase `send_response()`, koja za parametar prima upravo broj statusnog koda. Dakle, ako je zahtev uspešan, poslaće se kod 200.

Sledeće što HTTP server treba da upakuje u odgovor jeste zaglavlje. Zaglavlje se šalje metodom nasleđene BaseHTTPRequestHandler klase `send_header()`, koja za parametre prima dva elementa: ime polja zaglavlja i njegovu vrednost – jedno polje zaglavlja po jednom pozivu ove metode. Takođe, nakon što odredimo koja polja zaglavlja šaljemo, koristimo metodu `end_headers()` kako bismo naznačili gde je kraj čitavog zaglavlja, a odakle, opciono, počinje deo sa podacima koji se šalju. Tako, ako želimo da pošaljemo dva polja zaglavlja, učinimo to na sledeći način:

```
self.send_header('Content-type', 'text/plain')

self.send_header('Date', ' Date: Thu, 20 Aug 2020 12:00:00 GMT')

self.end_headers()
```

Na ovaj način, šaljemo dva polja u zaglavlju: `Content-type`, koji pokazuje da je tip podatka koji se šalje u ovom odgovoru tipa `text/plain`, i polje `Date`, koje pokazuje vreme i datum ovog odgovora.

Kako bismo iz upitnog stringa izvukli ime osobe za koju se traži prezime iz rečnika `names_dict`, moramo raščlaniti upitni string pomoću funkcije `parse_qs` ugrađenog modula `urllib.parse`. Pre svega, da bismo došli do upitnog stringa GET zahteva koji je klijent poslao, upotrebićemo `path` polje nasleđene BaseHTTPRequestHandler klase koje vraća string objekat. Pošto nam ono vraća ceo upitni string (`?name=michael`), a mi ne želimo da nam u ključ rečnika uđu karakteri `/` i `?`, koristićemo odsečak tog stringa:

```
parse_qs(self.path[2:])
```

Povratna vrednost funkcije `parse_qs` je rečnik, pa tako možemo imenu pristupiti naredbom:

```
name = parse_qs(self.path([2:]))['name'][0]
```

Na kraju koristimo nulti indeks iz razloga što `parse_qs` za dati ključ vraća vrednost tipa lista. Nakon ove procedure imamo ime koje je korisnik tražio i možemo dobiti odgovarajuće prezime koje mu pripada iz rečnika `names_dict` jednostavnim traženjem po ključu.

Konačno, pakovanje samih podataka u odgovor se vrši pomoću `wfile.write()` naredbe nasleđene BaseHTTPRequestHandler klase. Python, kao i drugi programski jezici, koristi analogiju između mrežnih konekcija i otvaranja fajlova. U neke fajl objekte možemo samo upisivati, a neke možemo samo čitati. Tako, `wfile` predstavlja konekciju između klijenta i servera u koju se može samo upisivati. Bilo koji podaci *upisani* u tu konekciju metodom `write()` se odmah šalju kao odgovor na klijentski zahtev. Parametar metode `write()` je tipa `bytes` i

zato, ako šaljem stringove, moramo koristiti string metodu `encode()`, koja će str objekat pretvoriti u bytes. Jedna takva naredba slanja odgovora prilagođena našem primeru izgleda ovako:

```
self.wfile.write(str(names_dict[name]).encode())
```

Ovaj proces implementacije GET zahteva se odnosi na slučaj da je zahtev uspešan, pa je zato potrebno i implementirati scenarije kada je zahtev neuspešan. Tačnije, u našem primeru, zahtev je neuspešan kada traženo ime ne postoji u rečniku i na takav zahtev treba vratiti statusni kod 404. Proveru uspešnosti zahteva možemo kodirati pomoću try-except bloka koda, gde se u slučaju dobavljanja prezimena na osnovu imena koje je podneo klijent, proverava da li to ime postoji u rečniku naredbom `if name in names_dict.keys()` i, u zavisnosti od vrednosti ove naredbe, klijentu šalje prezime za dato ime (statusni kod 200) ili vraća *Name not found* (statusni kod 404) u slučaju da ne postoji. Ista provera se odnosi i na poziv `parse_qs()` funkcije. U zavisnosti od toga da li nastane greška prilikom parsiranja ili ne, treba poslati statusni kod 404.

Ovim smo implementirali procesiranje GET metode/zahteva.

Metoda POST

Metoda POST se koristi kada klijent podnosi informacije ka serveru. U našem primeru, klijent će poslati novi par imena i prezimena serveru radi dodavanja tog para u naš rečnik. Implementaciju POST metode na našem serveru započinjemo definisanjem funkcije `def do_POST(self)`. Pre dodavanja ključ-vrednost para u naš rečnik, kao i kod GET metode, moramo na isti način raščlaniti upitni string koristeći `parse_qs` metodu čitajući string iz `self.path[2:]` naredbe. Pošto je reč o POST zahtevu, klijent šalje upitni string koji je deo URL-a čiji format izgleda ovako: `http://127.0.0.1:8080/?name=peter&last_name=peterson`. Ako je raščlanjivanje uspešno, novi ime-prezime par ćemo dodati komandom:

```
names_dict[data['name'][0]] = data['last_name'][0]
```

U slučaju da je klijent poslao drugačija polja u upitnom stringu – na primer, ako URL izgleda ovako: `http://127.0.0.1:8080/?nameS=peter&last_nameS=peterson`, naredba iz prethodnog reda će izbaciti `KeyError` grešku. U ovom slučaju, taj scenario treba iskodirati da se, ukoliko dođe do ove greške, klijentu pošalje 404 poruka sa greškom *Incorrect parameters provided* umesto poruke 200 sa novim, izmenjenim rečnikom. U oba slučaja, slanje odgovora klijentu se vrši na isti način kao i kod GET metode, i to pozivanjem sledećih metoda:

- `self.send_response()`
- `self.send_header()`
- `self.end_headers()`
- `self.wfile.write()`

Takođe, prilikom ove procedure važno je obratiti pažnju i na redosled njihovog pozivanja, jer će, ako se pozove prvo `self.send_header()` pa `self.send_response()`, doći do greške na serverskoj strani.

Radi efikasnijeg upravljanja greškama, a kako ne bismo ponavljali linije koda, dodaćemo pomoćnu funkciju našoj `RequestHandler` klasi: `def send_response_to_client(self, status_code, data)`, koja će biti zadužena samo za slanje statusnog koda, zaglavlja i tela

poruke. Ovo nije neophodno, ali olakšava čitanje i razumevanje koda. Ceo serverski kod izgleda ovako:

http_local_server.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import parse_qs
names_dict = {'john':'smith',
              'david':'jones',
              'michael':'johnson',
              'chris':'lee'}
class RequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.log_message("Incoming GET request...")
        try:
            name = parse_qs(self.path[2:])[ 'name' ][0]
        except:
            self.send_response_to_client(404, 'Incorrect parameters
provided')
            self.log_message("Incorrect parameters provided")
            return

        if name in names_dict.keys():
            self.send_response_to_client(200, names_dict[name])
        else:
            self.send_response_to_client(404, 'Name not found')
            self.log_message("Name not found")

    def do_POST(self):
        self.log_message('Incoming POST request...')
        data = parse_qs(self.path[2:])
        try:
            names_dict[data['name'][0]] = data['last_name'][0]
            self.send_response_to_client(200, names_dict)
        except KeyError:
            self.send_response_to_client(404, 'Incorrect parameters
provided')
            self.log_message("Incorrect parameters provided")

    def send_response_to_client(self, status_code, data):
        # Send OK status
        self.send_response(status_code)
        # Send headers
        self.send_header('Content-type', 'text/plain')
        self.end_headers()

        # Send the response
        self.wfile.write(str(data).encode())

server_address = ('127.0.0.1', 8080)
http_server = HTTPServer(server_address, RequestHandler)
http_server.serve_forever()
```

Napomena

U radu sa BaseHTTPRequestHandler klasom postoji još par metoda koje nismo pomenuli, a koje mogu biti korisne. Te metode su:

- `self.headers()` – vraća u program zaglavlje klijentskog zahteva;
- `self.log_message()` – evidentiranje poruka; koristi se umesto print naredbe;
- `self.address_string()` – vraća klijentsku adresu sa koje je zahtev stigao;
- `self.responses` – vraća rečnik gde su ključevi int vrednosti statusnih kodova, a vrednosti tih ključeva n-torke sa dva elementa gde je prvi element kraća a drugi element duža statusna poruka koja pripada datom statusnom kodu;
- `self.command` – vraća tip metode (GET ili PUT, ili POST itd).

Pitanje

Biblioteka na kojoj se bazira naša implementacija HTTP servera je:

- `http_server`
- **`http.server`**
- `httpserver`

Objašnjenje:

Biblioteka na kojoj se bazira naša implementacija HTTP servera se zove `http.server`.

HTTP klijent

Iako je HTTP klijent najčešće pregledač (browser), ne znači da nam je to jedina mogućnost pristupa HTTP serveru. Sve što može ostvariti TCP konekciju sa serverom nam može koristiti kao HTTP klijent. U našem primeru smo se opredelili za korišćenje biblioteke koja nije deo standardnog seta biblioteka – `requests`. Iako u Pythonu postoji biblioteka za rad sa HTTP protokolom – `urllib3`, biblioteka `requests` je izabrana zbog njene jednostavnosti i lakoće upotrebe prilikom kreiranja zahteva ka serveru. Pošto je ovo biblioteka napisana od strane zajednice, dakle nije deo ugrađenih biblioteka koje su došle prilikom instalacije Pythona, mora se instalirati koristeći `pip` naredbu iz komandnog prozora. Naime, za instalaciju `requests` naredbe u otvorenom komandnom prozoru treba pokrenuti sledeću komandu:

```
pip install requests
```

```
C:\WINDOWS\system32\cmd.exe
>pip install requests
Collecting requests
  Downloading https://files.pythonhosted.org/packages/45/1e/0c169c6a5381e241ba7404532c16a21d86ab872c9bed8bdc4c423954103/requests-2.24.0-py2.py3-none-any.whl (61kB)
    100% |#####| 71kB 995kB/s
Requirement already satisfied: certifi>=2017.4.17 in c:\python27\lib\site-packages (from requests) (2019.6.16)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\python27\lib\site-packages (from requests) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in c:\python27\lib\site-packages (from requests) (2.8)
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in c:\python27\lib\site-packages (from requests) (1.25.3)
Installing collected packages: requests
Successfully installed requests-2.24.0
You are using pip version 18.1, however version 20.2b1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

Slika 8.1. Prikaz uspešne instalacije requests biblioteke uz pomoć pip komande

Nakon uspešne instalacije ove biblioteke, potrebno ju je uvesti u našu skriptu naredbom `import requests`. Ova biblioteka nam omogućava vrlo jednostavan način za kreiranje GET i POST zahteva ka serveru:

- `r = requests.get(url)`
- `r = requests.post(url)`

Takođe, ostali zahtevi HTTP protokola se pomoću requests biblioteke pozivaju koristeći upravo njihova imena, i to na primer: `requests.put()`, `requests.head()`, `requests.delete()` itd. Više o ostalim metodama requests biblioteke se može pročitati na stranici [zvanične dokumentacije](#).

Na ovaj način smo kreirali requests objekat pomoću kojeg, kada stigne odgovor od servera, možemo pročitati taj odgovor, statusni kod, zaglavlja i ostalo.

Konkretno, u našem slučaju, pored URL-a koji je ovde <http://127.0.0.1:8080/>, moramo proslediti i upitne stringove. Upitni stringovi se u requests pozivu prosleđuju u vidu tipa rečnik koji se dodeljuju argumentu `params`:

- `r = requests.get("http://127.0.0.1:8080/", params={"name": 'michael'})`
- `r = requests.post("http://127.0.0.1:8080/", params = {'name': 'peter', 'last_name': 'peterson'})`

Ovo je sve što je potrebno da testiramo i GET i POST funkcionalnost našeg servera. Pa tako, kod klijentske strane(`http_local_client.py`) izgleda ovako:

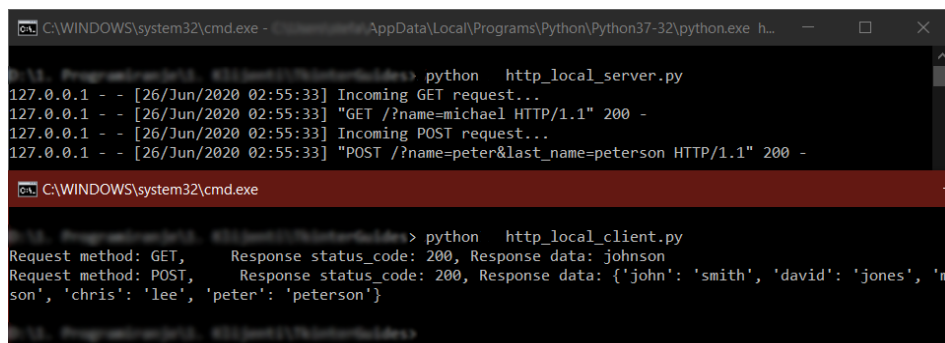
http_local_client.py

```
import requests
r = requests.get("http://127.0.0.1:8080/", params={"name": 'michael'})
print("Request method: GET, \
      Response status_code: {}, Response data: {}".format(r.status_code,
r.text))
r = requests.post("http://127.0.0.1:8080/", params = {'name': 'peter',
'last_name': 'peterson'})
print("Request method: POST, \
      Response status_code: {}, Response data: {}".format(r.status_code,
r.text))
```

Print naredbe nisu neophodne, ali su dodate radi lakšeg posmatranja rezultata.

Pokretanje i analiza rezultata

Za testiranje klijenta i servera potrebno je pokrenuti dva različita komandna prozora i pozicionirati se u direktorijumu gde se fajlovi `http_local_client.py` i `http_local_server.py` nalaze. Potrebno je prvo pokrenuti serversku stranu, a potom klijentsku skriptu.



```
C:\WINDOWS\system32\cmd.exe - AppData\Local\Programs\Python\Python37-32\python.exe h...
> python http_local_server.py
127.0.0.1 - - [26/Jun/2020 02:55:33] Incoming GET request...
127.0.0.1 - - [26/Jun/2020 02:55:33] "GET /?name=michael HTTP/1.1" 200 -
127.0.0.1 - - [26/Jun/2020 02:55:33] Incoming POST request...
127.0.0.1 - - [26/Jun/2020 02:55:33] "POST /?name=peter&last_name=peterson HTTP/1.1" 200 -

C:\WINDOWS\system32\cmd.exe
> python http_local_client.py
Request method: GET, Response status_code: 200, Response data: johnson
Request method: POST, Response status_code: 200, Response data: {'john': 'smith', 'david': 'jones', 'michael': 'peterson', 'chris': 'lee', 'peter': 'peterson'}
```

Slika 8.2. Prikaz pokretanja klijenta i servera kroz komandnu liniju

Prvo što primetimo kada pokrenemo serversku stranu je da se zapravo ništa ne dešava osim što server osluškuje nadolazeće zahteve na zadatom socketu. Čim pokrenemo klijentski fajl – server će početi da obrađuje zahteve. Na klijentskoj strani je prvi na redu GET zahtev sa upitnim stringom `name=michael`. Čim server potvrdi zahtev, ispisuje zaglavlje zahteva, koje u našem slučaju sadrži ime metode (GET), upitni string koji je deo URL-a, verziju HTTP protokola po kojoj se konekcija odvija (HTTP/1.1), kao i statusni kod. Nakon odbrade tog zahteva, server šalje klijentu odgovor koji se primećuje u donjem komandnom prozoru u formatu: Request method: GET, Response status_code: 200, Response data: johnson. Dakle, iz ovoga se primećuje da je server pronašao prezime za prosleđeno ime – johnson – i vratio taj podatak klijentu. U ovom trenutku se prekida konekcija između servera i klijenta, ali i započinje nova, ovog puta koristeći POST metodu. Klijent sada šalje POST zahtev za dodavanje novog para ime-prezime već postojećem rečniku. Server prepoznaje da je reč o POST zahtevu, vrši obradu tog zahteva i vraća novostvoreni rečnik kao dokaz o uspešnoj dopuni.

Ovo je bio prikaz uspešne komunikacije. Ako malo izmenimo klijentski kod tako da preko upitnog stringa šaljemo parametre koje server ne očekuje, dobićemo drugačiji rezultat.


```
C:\WINDOWS\system32\cmd.exe - C:\Users\user\AppData\Local\Programs\Python\Python37-32\python.exe h...
127.0.0.1 - - [26/Jun/2020 02:55:33] Incoming GET request...
127.0.0.1 - - [26/Jun/2020 02:55:33] "GET /?name=michael HTTP/1.1" 200 -
127.0.0.1 - - [26/Jun/2020 02:55:33] Incoming POST request...
127.0.0.1 - - [26/Jun/2020 02:55:33] "POST /?name=peter&last_name=peterson HTTP/1.1" 200 -
127.0.0.1 - - [26/Jun/2020 02:56:38] Incoming GET request...
127.0.0.1 - - [26/Jun/2020 02:56:38] "GET /?name=George HTTP/1.1" 400 -
127.0.0.1 - - [26/Jun/2020 02:56:38] Name not found
127.0.0.1 - - [26/Jun/2020 02:56:38] Incoming POST request...
127.0.0.1 - - [26/Jun/2020 02:56:38] "POST /?name=peter&last_name=peterson HTTP/1.1" 200 -

C:\WINDOWS\system32\cmd.exe
> python http_local_client.py
Request method: GET, Response status_code: 200, Response data: johnson
Request method: POST, Response status_code: 200, Response data: {'john': 'smith', 'david': 'jones', 'm
son', 'chris': 'lee', 'peter': 'peterson'}

> python http_local_client.py
Request method: GET, Response status_code: 400, Response data: Name not found
Request method: POST, Response status_code: 200, Response data: {'john': 'smith', 'david': 'jones', 'm
son', 'chris': 'lee', 'peter': 'peterson'}
```

Slika 8.3. Prikaz pokretanja klijenta i servera kroz komandnu liniju sa pogrešnim parametrima

U ovom slučaju smo se nadovezali na već postojeću sesiju servera iz prethodnog primera, i ponovo pokrenuli klijent, ali sada sa izmenjenim parametrima. Odmah uočavamo i koji su to parametri promenjeni. Reč je o GET zahtevu; preciznije, prosleđeno ime nije pronađeno u predefinisanoj rečniku. Server je prepoznao ovo i vratio statusni kod 400 sa porukom *Name not found*, što se može videti i iz klijentske komandne linije.

Napomena

Eksperimentisati sa slanjem različitih parametara od strane klijenta i analizirati te rezultate.

U ovoj nastavnoj jedinici smo realizovali HTTP klijent-server komunikaciju na primeru jednostavne manipulacije nad rečnikom. Međutim, HTTP protokol pruža mnogo više od toga, pa tako, uz manje dopune našeg programa, naš server umesto manipulacije rečnikom može vraćati string klijentu u vidu HTML koda. U tom slučaju klijent može biti i skripta sa requests bibliotekom, a i sam browser koji će taj HTML prikazati.

Rezime

- Za jednostavnu implementaciju HTTP servera koristimo ugrađenu biblioteku `http.server`.
- HTTP objekat dobijamo klasom `HTTPServer`, koja je deo `http.server` biblioteke.
- Konstruktor klase `HTTPServer` prihvata sledeće argumente: n-torku sa dva elementa (adresa i port) i klasu za obradu (handler klasu).
- Puštanje u rad objekta servera se vrši pomoću metode `server_forever()`.

- Klasa za obradu je klasa koju mi definišemo i implementiramo po svojim zahtevima, a koja zapravo nasleđuje BaseHTTPRequestHandler klasu koja je deo http.server biblioteke.
- BaseHTTPRequestHandler klasa u sebi već ima utvrđene mehanizme rukovanja sa HTTP metodama, koje mi samo proširujemo.
- Definisanje klasnih metoda koje odgovaraju metodama HTTP protokola se vrši po šablonu: do_IME_METODE.
- I klasna metoda GET i klasna metoda POST moraju generisati odgovor sledećim redosledom:
 - self.send_response()
 - self.send_header()
 - self.end_headers()
 - self.wfile.write()
- Za čitanje zahteva koji je došao preko upitnog stringa koristimo naredbu self.path.
- Za kreiranje klijentskog zahteva koristimo biblioteku requests, koja nije deo ugrađenih biblioteka Pythona. Uz pomoć ove biblioteke, za slanje GET zahteva koristimo naredbu requests.get(url, params), dok za slanje POST zahteva koristimo naredbu requests.post(url, params). U oba slučaja argument url je tipa string, a argument params je tipa rečnik.

